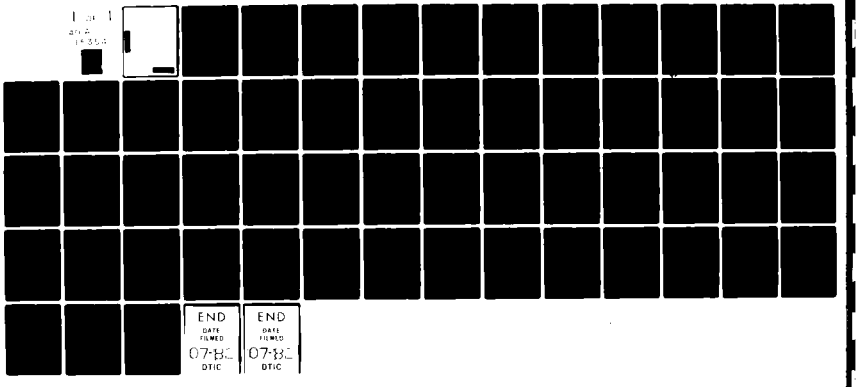
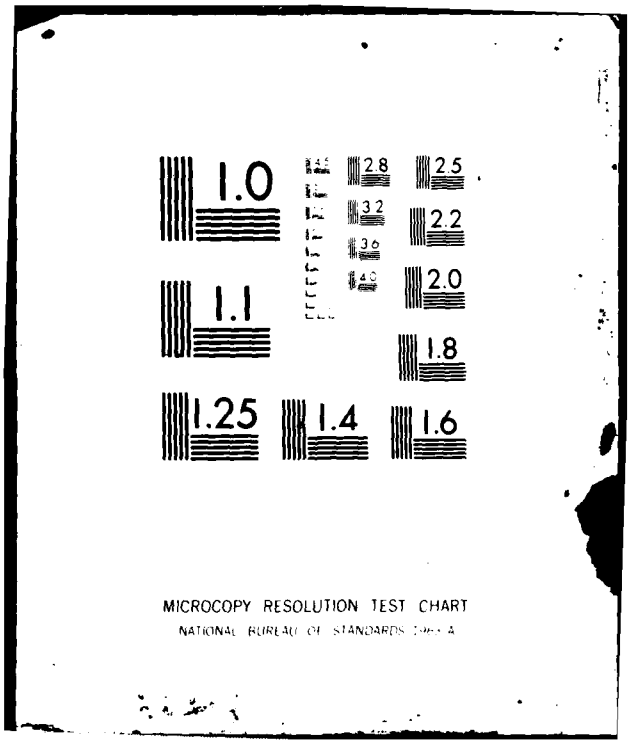
AD-A115 354 STANFORD UNIV CA F/6 12/1
CONSTRUCTING A MINIMAL COST SPANNING TREE SUBJECT TO RESOURCE C--ETC(U)
NOV 81 A W SHOGAN N00014-75-C-0561
UNCLASSIFIED TR-200 NL



END
DATE FILMED
07-82
DTIC

END
DATE FILMED
07-82
DTIC


1.0
1.1
1.25
1.4
1.6
2.8
3.2
3.6
4.0
2.5
2.2
2.0
1.8
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS

CONSTRUCTING A MINIMAL COST SPANNING TREE SUBJECT TO RESOURCE CONSTRAINTS AND FLOW REQUIREMENTS

by

Andrew W. Shogan

TECHNICAL REPORT NO. 200

November 1981

SUPPORTED UNDER CONTRACT N00014-75-C-0561 (NR-047-200)
WITH THE OFFICE OF NAVAL RESEARCH

Gerald J. Lieberman, Project Director

DEPARTMENT OF OPERATIONS RESEARCH
AND
DEPARTMENT OF STATISTICS
STANFORD UNIVERSITY
STANFORD, CALIFORNIA

CONSTRUCTING A MINIMAL COST SPANNING TREE SUBJECT TO RESOURCE CONSTRAINTS AND FLOW REQUIREMENTS

by

Andrew W. Shogan
University of California, Berkeley

1. INTRODUCTION

Given a connected, undirected, N-node network with a weight assigned to every arc, solving the minimal spanning tree (MST) problem requires the construction of a connected subnetwork that includes every node of the network and whose arcs have a minimal total weight. It is well-known that such a subnetwork will always be an (N-1)-arc tree, that is, a connected network with no cycles.

The construction of an MST is the goal in a wide variety of applications. For example, given a set of nodes that must communicate with each other and given the cost of constructing a communication link between each pair of nodes, the MST is the solution to the problem of constructing at minimal cost a network in which every pair of nodes can communicate along some path. A number of other seemingly unrelated problems and applications can be reduced to minimal spanning tree problems; both Bradley [1] and Kershenbaum and Van Slyke [19] provide excellent surveys.

There are two classic MST algorithms, one due to Kruskal [20] and another discovered independently by Prim [21] and Dijkstra[6]. Both algorithms consist of N-1 iterations, with each iteration identifying a new arc of an optimal spanning tree. An iteration of the Kruskal algorithm consists of adding an arc to a forest of subtrees; the added arc is the arc of least weight that when added to the forest of subtrees does not form a cycle. An iteration of the Prim-Dijkstra algorithm consists of adding an

arc to a single subtree; the added arc is the arc of least weight incident to a node in the subtree and a node not in the subtree. Both algorithms are efficient and easy to implement on a computer; Kershenbaum and Van Slyke [19], Yao [24], and Cheriton and Tarjan [4] discuss very efficient computer implementations that rely on sophisticated sorting and storage techniques.

In many practical situations, resource constraints and flow requirements preclude the construction of the unconstrained MST obtained by applying either the Kruskal or Prim-Dijkstra algorithm; this paper considers such a situation. More specifically, consider a set of N nodes having indices $1, 2, \ldots, N$; node 1 is a source having an infinite supply of a commodity, and every other node p is a sink having a known constant demand d_p. Denote the undirected arc between two nodes having indices $p < q$ by (p,q); note that there are $n \equiv (1/2)(N)(N-1)$ arcs. For notational convenience, assign to each arc a distinct index j equal to its position in a lexicographic ordering of the n arcs based on the representation of an arc as a vector (p,q) with $p < q$; that is, assign arc (p,q) the index $j = (1/2)(p-1)(2N-p) + (q-p)$, where $1 \leq p < q \leq N$, so that $1 \leq j \leq n$. The construction of the spanning tree requires the consumption of m scarce resources available in supplies b_i for $i = 1, 2, \ldots, m$. Associated with each potential arc j of the spanning tree are $m + 2$ known constants: c_j, the cost of constructing arc j, a_{ij} $(1 \leq i \leq m)$, the amount of resource i consumed during the construction of arc j, and e_j, the flow capacity of arc j. (Since there exists in any spanning tree a unique path between the source and a given sink, e_j is effectively an upper limit on the summation of demands over all

sinks whose unique paths to the source include arc j.) Given d_p for each sink p and c_j, a_{1j}, a_{2j}, ..., a_{mj}, e_j for each arc j, the problem is to construct a minimal cost spanning tree subject to the resource constraints (i.e., the consumption of each scarce resource cannot exceed its available supply) and subject to the requirement that there exists a feasible flow (i.e., a flow on the arcs of the spanning tree that satisfies the demands at the sinks without exceeding any arc capacity). Hereafter, this problem will be referred to as the resource-constrained, capacitated minimal spanning tree problem and abbreviated as the RCMST problem.

Note the following characteristics of the RCMST problem:

(1) It is assumed that between any pair of nodes there exists only one arc under consideration for inclusion in the spanning tree; Section 5 relaxes this assumption in order to consider the case where any one of multiple arcs between a pair of nodes is eligible for inclusion in the spanning tree.

(2) If for some arc j, $c_j = \infty$, $a_{ij} > b_i$ for some i, or $e_j = 0$, then arc j may be excluded from consideration.

(3) The constants c_j, a_{ij} for $i = 1, 2, \ldots, m$, and e_j for some arc j may be interrelated; for example, a portion of the construction cost c_j may be attributable to the costs of the scarce resources consumed and/or the magnitude of the capacity or, for example, one of the resource constraints may limit the number of arcs of a given capacity that may be included in the spanning tree.

(4) Each resource constraint need not represent a resource in the strict sense of the word. For example, by setting $a_{ij} = 1$

if arc j is incident to node i and $a_{ij} = 0$ otherwise, the i-th resource constraint may be used to require the degree of node i in the spanning tree to be no greater than some constant b_i; hereafter, such a resource constraint will be referred to as a "degree constraint".

(5) If the demands d_p, $2 \leq p \leq N$, all equal 1, then an arc capacity e_j $(1 \leq j \leq n)$ may be interpreted as the maximum number of nodes that may access the source through arc j if it were to be included in the spanning tree. For example, $e_j = K$ for any arc j incident to the source and $e_j = \infty$ otherwise models the situation where at most K nodes (e.g., computer terminals) may be contained in any subtree connected directly to the source (e.g., a central computer) by arc j (e.g., a port). As another example, $e_j = K$ for every arc j models a "reliability constraint" stating that at most K demand nodes may be disconnected from the sink by the removal (failure) of any arc from the spanning tree.

(6) Requiring the constructed network to be a tree implies that there are reasons (perhaps physical or technological) that prevent the flow from the source to a particular sink to be transmitted simultaneously along more than one path.

In its most general form, the RCMST problem has not yet been discussed in the literature; neither has the special case, hereafter referred to as the Resource-constrained Minimal Spanning Tree (RMST) problem, in which $e_j = \infty$ for every arc j (i.e., the flow constraints may be ignored so that only the resource constraints are present). However, two

special cases of the RCMST problems have received significant attention by other researchers: (1) the Degree-constrained Minimal Spanning Tree (DMST) problem in which $e_j = \infty$ for every arc j and in which all resource constraints are "degree constraints" as described earlier and (2) the Capacitated Minimal Spanning Tree (CMST) problem in which $b_i = \infty$ for every resource i (i.e., the resource constraints may be ignored so that only the flow constraints are present). Both the DMST and the CMST problems are known to be NP-hard; consequently, most approaches to these problems have been heuristic rather than exact. Kershenbaum [18] has surveyed and analyzed the computational complexity of a class of heuristic algorithms for the CMST problem; Chandy and Lo [3] have proposed a branch-and-bound algorithm for obtaining the exact solution to the CMST problem; and, more recently, Gavish [11, 12] has proposed a Lagrangean-based approach to both the DMST and CMST problems.

The remainder of this paper is organized as follows. Section 2 discusses applications of the RCMST problem; Section 3 describes a branch-and-bound algorithm (based on Lagrangean relaxation) for solving the RMST problem and illustrates the algorithm with an example; Section 4 discusses the modifications to the algorithm necessary to solve the more general RCMST problem; Section 5 discusses further modifications necessary to solve two extensions of the RCMST problem, the existence of multiple arcs between a pair of nodes and the existence of multiple sources; finally, Section 6 reports some preliminary computational experience with a computer implementation of the algorithm.

In the remainder of this paper, the following conventions, notation, and terminology will apply:

(1) all vectors are row vectors;

(2) c will denote a vector whose j-th component is c_j, the cost of constructing arc j;

(3) whenever "0" is used where a vector would normally appear, it represents a vector with every component equal to 0;

(4) if $x = (x_1, x_2, \ldots, x_n)$, then $\|x\| = \sum_{i=1}^{n} x_i^2$;

(5) if x and y are two vectors having the same dimension, xy represents the product of x and y, where it is implicitly assumed that y has been transposed so that the product is defined;

(6) if x and y are two vectors having the same dimension, max (x,y) represents a vector whose i-th component is the maximum of the i-th component of x and the i-th component of y;

(7) if A and B are two sets, A-B denotes the intersection of A and the complement of B, that is, the set of elements in A but not in B;

(8) the existence of a "feasible flow" in a tree implies that there exists a flow on the arcs of the tree such that the flow on any arc does not exceed the arc's capacity and the difference between the flow out of a node and the flow into a node equals the node's supply if it is a source, the negative of the node's demand if it is a sink, and zero otherwise.

2. APPLICATIONS

The RCMST problem has applications to the design and construction of a tree network in which a set of demand nodes and a single source must be interconnected by selecting from a set of potential arcs, each having an associated capacity and resource consumptions. For example, the demand nodes may be computer terminals, the source may be a central computer, and the arcs may be potential transmission lines (cf. [3], [12], and [18]); alternatively, the demand nodes may be offshore natural gas fields, the source may be an onshore separation and compression plant, and the arcs may be potential pipelines (cf. [22]).

Although the RCMST problem has applications to the design and construction of spanning trees, the primary motivation for consideration of the RCMST problem was the problem of reconstructing a network after a natural disaster, such as an earthquake. Cities, states, and countries rely heavily on a variety of networks: energy networks (such as electrical or natural-gas networks), communication networks (such as telephone, telegraph, or computer networks), transportation networks (such as highway or railroad networks), and water networks (such as networks for the distribution of potable water or the removal of sewage). Using an analogy with the human body's life-supporting "networks" (e.g., the vascular and neurological systems), C.M. Duke and D. F. Moran [7] coined the term "lifelines" to refer to such networks, and their analysis and recommendations following the 1971 San Fernando (California) earthquake lead to an increased emphasis by professionals and academicians on the problems of "lifeline earthquake engineering". In the last 10 years, much research has been devoted to predicting the level of damage a particular lifeline might

sustain in an earthquake and to designing new lifelines and "retrofitting" existing lifelines to limit the potential damage as much as possible. However, not nearly as much effort has been devoted to the decision problems associated with optimally restoring the services provided by a lifeline that has been damaged by an earthquake. Restoration of service is an important problem if one accepts the fact that in certain regions of the world, earthquakes and the resulting damage to lifelines are inevitable.

Many important lifelines do not have a tree structure, and, after sustaining damage, the long-run goal for such a network will be to rebuild the network so that it is as good or better than it was prior to the earthquake. However, the short-run goal may be less ambitious. In the short-run, the goal may simply be to allocate scarce resources (labor, equipment, spare parts, etc.) to the repair process in such a way as to restore some "minimal level of service" as quickly and economically as possible. One possible interpretation of "minimal level of service" is that the short-run goal would be to construct a spanning tree (when only a single source exists) or a spanning forest (when multiple sources exist) so that every demand node would have access to a source. Stated more formally in the context of the RCMST problem, suppose that the short-run post-earthquake goal for a damaged network is to construct a minimal cost spanning tree by repairing a sufficient number of arcs. The restoration of arc j $(1 \leq j \leq n)$ to its former capacity e_j would cost c_j dollars and would consume a_{ij} $(1 \leq i \leq m)$ units of the limited quantity of b_i units of resource i. The extensions discussed in Section 5 are applicable if it is possible to repair an arc in a variety of ways, with each way having a distinct associated cost, capacity, and consumption of resources and/or if there exist multiple sources.

3. SOLVING THE RMST PROBLEM

Let x denote a binary vector whose j-th component x_j equals 1 if arc j is included in the spanning tree and equals 0 otherwise; hereafter, such an x will be referred to as an incidence vector. Edmonds [8] has shown that x is an incidence vector corresponding to a tree if and only if x is an extreme point of the feasible region of a finite but very large system of linear inequalities; hereafter, $x \in T$ will denote that $x_j = 0$ or 1 for $1 \leq j \leq n$ and x also satisfies Edmonds' system of linear inequalities, or equivalently, x is an incidence vector corresponding to a tree.

The RMST problem can now be formulated as the following binary integer linear program:

$$
\begin{aligned}
\text{(P)} \qquad & \text{Minimize} \quad \sum_{j=1}^{n} c_j x_j \\
& \text{subject to} \quad \sum_{j=1}^{n} a_{ij} x_j \leq b_i \quad \text{for} \quad 1 \leq i \leq m, \\
& \qquad \text{and} \quad x \in T.
\end{aligned}
$$

Were it not for the m resource constraints, (P) would simply be a classic MST problem and could be easily solved by either of the algorithms discussed in Section 1. However, the presence of the resource constraints precludes an easy solution of (P).

Using one of the binary integer linear programming algorithms to solve (P) is impractical due to the unmanageable size of the system of linear inequalities represented by $x \in T$. Instead, the approach taken in

this paper will be to solve (P) by a branch-and-bound algorithm based on the technique of Lagrangean relaxation.

Fisher [9], Shapiro [23], Fisher, Northup, and Shapiro [10], and Geoffrion [13] have all written excellent surveys of Lagrangean relaxation. The technique was applied first in 1970 and 1971 by Held and Karp [14, 15] in their highly successful branch-and-bound approach to the traveling salesman problem. Held and Karp used a variant of a tree called a 1-tree, defined as a network consisting of N nodes and N arcs and obtained by connecting node 1 to any two nodes in a spanning tree interconnecting nodes 2, 3, ..., N. Thus, a 1-tree always has one and only one cycle, this cycle always contains node 1, and node 1 always has degree two. Also, note that a minimum-cost 1-tree can be efficiently found by constructing a minimum-cost spanning tree interconnecting the nodes 2, 3, ..., N and then connecting node 1 to the tree by the two arcs incident to node 1 having the lowest cost. Upon observing that a tour for the traveling salesman is simply a 1-tree in which each node has degree two, Held and Karp formulated the traveling salesman problem as the minimum-cost 1-tree problem subject to the additional constraints that each node has degree two. The formulation is similar to that of (P) with T modified slightly to take into account that x should correspond to a 1-tree and the resource constraints replaced by linear equations requiring each node to have degree two. Held and Karp solved their degree-constrained minimum-cost 1-tree problem by developing a branch-and-bound algorithm that for the first time used the concept of Lagrangean relaxation. Subsequent to the pioneering work of Held and Karp,

many applications of Lagrangean relaxation have appeared in the literature. Fisher [9] provides an excellent survey and a comprehensive bibliography.

The discussion to follow of the branch-and-bound algorithm used to solve (P) will employ the standard terminology (cf. [17, pp. 716-718). In particular, at any point in the algorithm, let Z_I denote the objective value of the incumbent, the best known feasible solution of (P); unless a feasible solution is known at the start, Z_I is initially set to ∞.

It is helpful to think of the algorithm as producing a rooted tree (hereafter referred to as the b&b-tree in order to distinguish it from a spanning tree), where the root of the b&b-tree corresponds to problem (P). Every other node of the b&b-tree corresponds to a problem similar to (P) but having added restrictions placed on the incidence vector x. In particular, associated with each node of the b&b-tree are two mutually exclusive subsets A and B of the set of arc indices $\{1, 2, \ldots, n\}$; the subset A consists of indices of arcs that not only form a subtree but also must be "admitted" to the spanning tree ($x_j = 1$ for $j \in A$), and the subset B consists of indices of arcs that must be "banished" from the spanning tree ($x_j = 0$ for $j \in B$). Given its associated A and B, a node of the b&b-tree corresponds to the binary integer linear program

$$(P_{AB}) \qquad \text{Minimize} \quad \sum_{j=1}^{n} c_j x_j$$

$$\text{subject to} \quad \sum_{j=1}^{n} a_{ij} x_j \leq b_i \quad \text{for} \quad 1 \leq i \leq m \ ,$$

$$\text{and} \quad x \in T_{AB} \ ,$$

where $T_{AB} = T \cap \{x | x_j = 1 \text{ for } j \in A\} \cap \{x | x_j = 0 \text{ for } j \in B\}$. Because $T_{AB} \subseteq T$, problem (P_{AB}) will be referred to as a "subproblem" of problem (P). At any stage of the branch-and-bound algorithm, the T_{AB} sets associated with the leaves of the b&b-tree are a partition of T; in this sense, then, the subproblems (P_{AB}) associated with the leaves of the b&b-tree are a "partition" of the problem (P).

The Lagrangean relaxation (L-relaxation hereafter) of problem (P_{AB}) relative to the m resource constraints and a given nonnegative dual vector $\lambda = (\lambda_1, \lambda_2, \ldots, \lambda_m)$ is the binary integer linear program

$$(P_{AB}^{\lambda}) \qquad \text{Minimize} \quad \sum_{j=1}^{n} c_j x_j + \sum_{i=1}^{m} \lambda_i \left(\sum_{j=1}^{n} a_{ij} x_j - b_i \right)$$

$$\text{subject to} \quad x \in T_{AB},$$

or, equivalently,

$$\text{Minimize} \quad \sum_{j=1}^{n} \left(c_j + \sum_{i=1}^{m} \lambda_i a_{ij} \right) x_j - \sum_{i=1}^{m} \lambda_i b_i$$

$$\text{subject to} \quad x \in T_{AB} \ .$$

Note that for a given nonnegative λ, (P_{AB}^{λ}) is equivalent to an MST problem where the weight associated with arc j is $-\infty$ if $j \in A$, ∞ if $j \in B$, and $\left(c_j + \sum_{i=1}^{m} \lambda_i a_{ij}\right)$ otherwise. As such, (P_{AB}^{λ}) may be solved easily by any of the MST algorithms discussed in Section 1.

For a given A and B and any nonnegative λ, let v_{AB}^{*} and $w_{AB}^{*}(\lambda)$ denote the optimal objective values for problems (P_{AB}) and (P_{AB}^{λ}), respectively. Clearly, $w_{AB}^{*}(\lambda) \leq v_{AB}^{*}$; that is, $w_{AB}^{*}(\lambda)$ is a lower bound on v_{AB}^{*} for any nonnegative λ. A traditional branch-and-bound approach might use $w_{AB}^{*}(0)$ as a lower bound for v_{AB}^{*}, that is, compute a

lower bound on v^*_{AB} by simply ignoring the resource constraints and solving the resulting minimal spanning tree problem. However, a branch-and-bound algorithm based on L-relaxation seeks to increase the likelihood of fathoming a subproblem (P_{AB}) by searching for the greatest possible lower bound by considering the following dual problem of (P_{AB}).

$$(D_{AB}) \qquad \text{maximize} \quad w^*_{AB}(\lambda)$$

$$\text{subject to} \quad \lambda_i \geq 0 \quad \text{for} \quad 1 \leq i \leq m \quad .$$

Note that although problem (D_{AB}) will be referred to as the dual of problem (P_{AB}), a "duality gap" (cf. [10, p. 57] or [13, p. 87]) may be present; that is, the optimal objective value for (D_{AB}) may be strictly less than the optimal objective value for (P_{AB}). Also, as indicated in the discussion below of the branch-and-bound algorithm, there are other potential benefits to considering (D_{AB}) in addition to an increased likelihood of fathoming a subproblem (P_{AB}) based on a lower bound on its optimal objective value. In particular, instead of simply solving (P^0_{AB}) as a more traditional approach might do, the attempt to solve (D_{AB}) involves the solution of a series of problems of the form (P^λ_{AB}) with a different value of λ each time; this increases the likelihood of fathoming the subproblem (P_{AB}) because each solution of a particular (P^λ_{AB}) may result in one or more of the following possibilities:

(1) an increased lower bound on the optimal objective value of (P_{AB}),

(2) a decrease in the objective value of the incumbent,

(3) the optimal solution to (P_{AB}).

Note that possibility (2) also may lead to the fathoming of other subproblems besides the particular (P_{AB}) under consideration.

Figure 1 contains a flow chart of the branch-and-bound algorithm; a more detailed discussion of each of the steps follows. As indicated earlier, for a given A and B and any nonnegative λ, $w^*_{AB}(\lambda)$ denotes the optimal objective value for problems (P^{λ}_{AB}); also, let $x^*_{AB}(\lambda)$ denote an optimal incidence vector for (P^{λ}_{AB}) and let $s^*_{AB}(\lambda)$ denote an m-vector whose i-th component is the value of $b_i - \sum_{j=1}^{n} a_{ij}x_j$ evaluated at $x = x^*_{AB}(\lambda)$, that is, the value of the slack variable for the i-th resource constraint when evaluated at the optimal solution to (P^{λ}_{AB}).

Steps 1, 2, and 10: The stack is a list of unfathomed subproblems. A subproblem (P_{AB}) on the stack is characterized by a "vector of information" $[A,B,\lambda,w^*_{AB}(\lambda),s^*_{AB}(\lambda)]$, where λ is not necessarily the optimal solution to problem (D_{AB}) but is such that $w^*_{AB}(\lambda)$ is the greatest known lower bound on v^*_{AB}. Initially (Step 1), the stack contains a single subproblem $[\phi,\phi,0,w^*_{\phi\phi}(0),s^*_{\phi\phi}(0)]$ corresponding to solving the minimal spanning tree problem $(P^{0}_{\phi\phi})$ that results from ignoring the resource constraints in problem $(P_{\phi\phi}) = (P)$. When the stack contains more than one item, it is helpful to think of items on the stack as being ordered from the top to bottom in increasing magnitudes of the lower bound $w^*_{AB}(\lambda)$. In selecting a new subproblem from the stack (Step 2), the chosen subproblem is the one with the lowest lower bound $w^*_{AB}(\lambda)$, that is, the subproblem on the top of the stack. Of course, if the stack is empty (Step 10), all subproblems have been fathomed and the incumbent incidence vector

is the optimal solution to (P); actually, since it is not unusual for the incumbent to change after a subproblem has been placed on the stack, a non-empty stack should be emptied by the fathoming of the entire stack whenever the item on the top of the stack has a lower bound $w^*_{AB}(\lambda)$ greater than or equal to the incumbent's objective value.

Overview of Steps 3-6: Upon the selection of a subproblem (P_{AB}) characterized by the "vector of information" $[A,B,\lambda,w^*_{AB}(\lambda), s^*_{AB}(\lambda)]$, an attempt is made to solve the problem (D_{AB}). Because of the work of Edmonds [8] discussed earlier in this section, problem (P^{λ}_{AB}) has the Integrality Property of Geoffrion [13, p. 89]; hence, the optimal solution to problem (D_{AB}) may be obtained in principle as the optimal dual variables associated with the resource constraints of the linear program that results from replacing "$x_j = 0$ or 1" with "$0 \leq x_j \leq 1$" $(1 \leq j \leq n)$ in the binary integer linear program (P_{AB}). However, solving (D_{AB}) in such a manner is impractical due to the large number of linear inequalities represented in (P_{AB}) by $x \in T_{AB}$; hence, an attempt to solve (D_{AB}) will be made through an iterative procedure (Steps 3-6) known as subgradient optimization (cf. [9], [10], [13], and [16]). Subgradient optimization is so named because $-s^*_{AB}(\lambda)$ is a subgradient of the objective function for (D_{AB}) at the point λ, and the procedure optimistically uses this subgradient as if it were a gradient pointing in a direction of steepest ascent. Iteration k of subgradient optimization consists of the following steps:
(3) movement to a new dual vector λ^k obtained recursively from λ^{k-1}, (4) solution of the L-relaxation $(P^{\lambda^k}_{AB})$, (5) attempting to fathom the

subproblem (P_{AB}) based on the results of solving $(P_{AB}^{\lambda^k})$, and (6) deciding whether to perform another iteration if the fathoming attempt was unsuccessful. The hope is that the sequence $\{\lambda^k\}$ will yield a good suboptimal, if not an optimal solution of (D_{AB}). Each step of subgradient optimization will now be discussed in detail.

Step 3: The revision of the dual vector λ is the most complicated procedure in the branch-and-bound algorithm. The procedure is based on a proof (omitted here but contained in [15] and [16]) that if the nonnegative scalar θ_k is sufficiently small, then

$$\lambda^k \equiv \max [0, \lambda^{k-1} - \theta_k \, s^*_{AB}(\lambda^{k-1})]$$

is closer than λ^{k-1} (as measured by Euclidean distance) to the optimal solution of (D_{AB}), although $w^*_{AB}(\lambda^k) \geq w^*_{AB}(\lambda^{k-1})$ need not hold. The complication, then, lies in the choice of θ_k; Held and Karp [15] successfully used $\theta_k = 1$ for all k in their work on the traveling salesman problem, and Held, Wolfe, and Crowder [16] later validated a more general method for choosing a value for θ_k. (Those readers not interested in further detail on the procedure for revising λ should skip to the discussion of Step 4.) Actually, the procedure used for revising λ in the computer implementation of the branch-and-bound algorithm discussed in Section 6 is a generalization of the above expression for λ^k that is due to Camerini, Fratta, and Maffioli [2] and Crowder [5]. For completeness, the procedure is now summarized without justification. Given a subproblem (P_{AB}) characterized by the "vector of information"

$[A,B,\lambda,w^*_{AB}(\lambda),s^*_{AB}(\lambda)]$, set $\lambda^0 = \lambda$ and set the m-vector $d^0 = 0$; the sequence of dual vectors $\{\lambda^k\}$ is generated by the recursions

$$d^k = - s^*_{AB}(\lambda^{k-1}) + \beta d^{k-1}$$

$$\theta_k = \begin{cases} 1 & \text{if } Z_I = \infty \\ \alpha_k \dfrac{Z_I - w^*_{AB}(\lambda^{k-1})}{\|s^*_{AB}(\lambda^{k-1})\|^2} & \text{if } Z_I < \infty \end{cases}$$

$$\lambda^k = \max [0, \lambda^{k-1} + \theta_k d^k] \quad ,$$

where β is a known nonnegative constant ($\beta = 0.6$ in the computational experience discussed in Section 6) and $\{\alpha_k\}$ is a predetermined sequence of nonnegative scalars generated according to the following policy (c.f. [9, p. 8], [16, p. 68] and [5, p. 361]): (1) set $\alpha_k = 2$ for $2N$ iterations, where N is the number of nodes in the network, (2) then successively halve both the value of α_k and the number of iterations α_k remains constant until the number of iterations is less than 5, (3) thereafter, α_k is halved every 5 iterations. Note that the above expression for d^k can be expanded to yield

$$d^k = - s^*_{AB}(\lambda^{k-1}) - \beta s^*_{AB}(\lambda^{k-2}) - \beta^2 s^*_{AB}(\lambda^{k-3}) - \cdots - \beta^{k-1} s^*_{AB}(\lambda^0)$$

so that the direction of movement away from λ^{k-1} is a discounted composite of the current and previous slack vectors (subgradients).

Step 4: Given the revised dual vector λ, (P_{AB}^{λ}) is equivalent to an MST problem where the weight associated with arc j is $-\infty$ if $j \in A$, ∞ if $j \in B$, and $\left(c_j + \sum_{i=1}^{m} \lambda_i a_{ij}\right)$ otherwise. As discussed in Step 7 below, the arcs whose indices are in A form a subtree. Hence, (P_{AB}^{λ}) may be solved efficiently by first forming the subtree of "admitted" arcs and then continuing with the Prim-Dijkstra algorithm as described by Prim [21].

Step 5: Upon solving (P_{AB}^{λ}), an attempt is made to fathom the subproblem (P_{AB}), that is, to remove the subproblem from further consideration. As indicated by the flowchart contained in Figure 2, the fathoming subroutine consists of the four steps A, B, C, and D. In Step A, a check is made on whether $(1+\varepsilon)\, w_{AB}^{*}(\lambda) \geq Z_I$ holds, where 100ε is the given percentage error tolerance ($\varepsilon = 0.00$, 0.01, or 0.05 in the computational experience discussed in Section 6). With $w_{AB}^{*}(\lambda)$ being a lower bound on v_{AB}^{*}, if $(1+\varepsilon) w_{AB}^{*}(\lambda) \geq Z_I$ holds, then the subproblem can be fathomed since 100ε is the maximum percentage error that could result if the optimal incidence vector for problem (P) belonged to the feasible region of (P_{AB}). If $(1+\varepsilon) w_{AB}^{*}(\lambda) < Z_I$, the fathoming subroutine proceeds to Step B where a check is made on whether $x_{AB}^{*}(\lambda)$ is a feasible for (P_{AB}) and therefore (P), that is, on whether the minimal spanning tree for (P_{AB}^{λ}) also satisfies the resource constraints of (P). The feasibility check is easily made by checking for a strictly negative component in the slack vector $s_{AB}^{*}(\lambda)$; if $s_{AB}^{*}(\lambda) \geq 0$ does <u>not</u> hold, the fathoming subroutine ends with (P_{AB}) unfathomed; if $s_{AB}^{*}(\lambda) \geq 0$, the fathoming subroutine proceeds to Step C in order to update the incumbent if necessary. More specifically,

$$w^*_{AB}(\lambda) = cx^*_{AB}(\lambda) - \lambda s^*_{AB}(\lambda) < Z_I$$

holds since (P_{AB}) was not fathomed in Step A; however,

$$cx^*_{AB}(\lambda) = w^*_{AB}(\lambda) + \lambda s^*_{AB}(\lambda) < Z_I$$

need not hold. Given the verification in Step B of the feasibility of $x^*_{AB}(\lambda)$ for problem (P), if $cx^*_{AB}(\lambda) < Z_I$ does hold, then $x^*_{AB}(\lambda)$ becomes the new incumbent; otherwise, the incumbent remains unchanged. Regardless of the outcome of Step C, Step D makes another attempt to fathom (P_{AB}). In particular,

$$w^*_{AB}(\lambda) = cx^*_{AB}(\lambda) - \lambda s^*_{AB}(\lambda) \leq v^*_{AB} \leq cx^*_{AB}(\lambda) \quad ,$$

where the last inequality follows from the verification in Step B of the feasibility of $x^*_{AB}(\lambda)$ for problem (P_{AB}); hence, if complementary slackness holds, that is, if $\lambda s^*_{AB}(\lambda) = 0$, then $w^*_{AB}(\lambda) = v^*_{AB} = cx^*_{AB}(\lambda)$ so that λ and $x^*_{AB}(\lambda)$ are optimal solutions to problems (D_{AB}) and (P_{AB}), respectively. Actually, if $\lambda s^*_{AB}(\lambda) \leq \varepsilon\, w^*_{AB}(\lambda)$, subproblem (P_{AB}) can be fathomed since 100ε is the maximum percentage error that could result by assuming $x^*_{AB}(\lambda)$ is the optimal solution to (P_{AB}). Note that if subproblem (P_{AB}) is fathomed in Step D, then $x^*_{AB}(\lambda)$ will always have just become the new incumbent in Step C since

$$cx^*_{AB}(\lambda) = w^*_{AB}(\lambda) + \lambda s^*_{AB}(\lambda) \leq (1+\varepsilon) w^*_{AB}(\lambda) < Z_I \quad ,$$

where the first inequality follows because $\lambda s^*_{AB}(\lambda) \leq \varepsilon\, w^*_{AB}(\lambda)$ and the second inequality follows from the fact that (P_{AB}) was not fathomed in Step A.

Step 6: If the subproblem (P_{AB}) was not fathomed in Step 5, a decision must be made as to whether to continue with subgradient optimization. Let $\lambda^0, \lambda^1, \lambda^2, \ldots, \lambda^k$ denote the dual vectors generated during the first k iterations of subgradient optimization. Recall that $w^*_{AB}(\lambda^0), w^*_{AB}(\lambda^1), w^*_{AB}(\lambda^2), \ldots, w^*_{AB}(\lambda^k)$ need not be a nondecreasing sequence; hence, it is necessary during subgradient optimization to initialize and update an incumbent dual vector λ^I, that is, that vector among $\lambda^0, \lambda^1, \lambda^2, \ldots, \lambda^k$ that produces the greatest lower bound on v^*_{AB}. A $(k+1)$-st iteration of subgradient optimization is performed unless one of the following two situations occur: (1) $k > 4N$, where N is the number of nodes in the network, or (2) within the 5 most recent iterations, $w^*_{AB}(\lambda^I)$ has not increased by at least 0.1%.

Steps 7, 8 and 9: After an unsuccessful attempt to fathom the subproblem via subgradient optimization, (P_{AB}) must be partitioned into two subproblems. Refer to the set of arcs whose indices are in A as the "A-arcs". Assume inductively that (if $A \neq \phi$) the A-arcs form a subtree and that at least one of the arcs of the subtree is incident to the source; hereafter, this subtree will be referred to as the A-subtree. If $A \neq \phi$, let J_{AB} denote the set of arc indices such that $j \in J_{AB}$ if and only if $j \notin B$ and one of the two nodes which arc j is incident is a member of the A-subtree and the other node is not; if $A = \phi$, let J_{AB} equal the

indices of those arcs not in B but incident to the source. Also, let j^* denote the index that minimizes $c_j + \sum_{i=1}^{m} \lambda_i^I a_{ij}$ over $j \in J_{AB}$ (recall λ^I is the incumbent dual vector after subgradient optimization). Then, partition (P_{AB}) into the two subproblems $(P_{A\cup\{j^*\},B})$ and $(P_{A,B\cup\{j^*\}})$. Note from the inductive hypothesis and the definition of j^*, that the $A\cup\{j^*\}$-arcs form a subtree and at least one of the arcs in the subtree is incident to the source. Note also that j^* is simply the index of the arc that connects the subtree corresponding to A (node 1 if $A = \phi$) to its closest neighbor as measured by the objective function coefficients of $\{x_j \mid j \in J_{AB}\}$ in problem $(P_{AB}^{\lambda^I})$; that is, j^* was the index of the first arc added to the A-subtree when the Prim-Dijkstra algorithm was used during subgradient optimization to solve the MST problem $(P_{AB}^{\lambda^I})$. Consequently, $(P_{A\cup\{j^*\},B}^{\lambda^I})$ and $(P_{AB}^{\lambda^I})$ have the same optimal solution so that $(P_{A\cup\{j^*\},B})$, in the form of the "vector of information" $[A\cup\{j^*\},B,\lambda^I,w_{AB}^*(\lambda^I),s_{AB}^*(\lambda^I)]$, should be added to the stack of unfathomed subproblems unless fathoming is possible due to the occurrence of one of the following situations:

(1) The amount of a particular resource used by the "admitted" $A\cup\{j^*\}$-subtree already exceeds the supply available for the entire spanning tree; that is, $\sum_{j\in A\cup\{j^*\}} a_{ij} > b_i$. In such a case, $(P_{A\cup\{j^*\},B})$ can be fathomed because it has no feasible solution.

(2) Situation (1) has not occurred but $A\cup\{j^*\}$ contains $N-1$ indices, where N is the number of nodes in the network. In such a case, $(P_{A\cup\{j^*\},B})$ can be fathomed because its only feasible solution, and therefore its optimal solution, is known, namely the spanning tree formed by the $N-1$ arcs corresponding to $A\cup\{j^*\}$.

The subproblem $(P_{A,B\cup\{j^*\}})$ cannot be dealt with as easily and must be treated as follows:.

(1) A lower bound on $v^*_{A,B\cup\{j^*\}}$ is obtained by solving (as discussed in Step 4) the MST problem $(P^{\lambda^I}_{A,B\cup\{j^*\}})$.

(2) Using the resulting $w^*_{A,B\cup\{j^*\}}(\lambda^I)$, $x^*_{A,B\cup\{j^*\}}(\lambda^I)$, and $s^*_{A,B\cup\{j^*\}}(\lambda^I)$, an attempt is made to fathom $(P_{A,B\cup\{j^*\}})$ by applying the fathoming subroutine as described in Step 5; note $w^*_{A,B\cup\{j^*\}}(\lambda^I)$ is always greater than or equal to $w^*_{AB}(\lambda^I)$ and may even equal ∞ if $B\cup\{j^*\}$ contains many arc indices.

(3) If $(P_{A,B\cup\{j^*\}})$ cannot be fathomed, it is added to the list of unfathomed subproblems in the form of the "vector of information"

$$[A,B\cup\{j^*\},\lambda^I,w^*_{A,B\cup\{j^*\}}(\lambda^I),s^*_{A,B\cup\{j^*\}}(\lambda^I)].$$

To illustrate the branch-and-bound algorithm discussed in the previous section, consider the following example having 6 nodes, 15 potential arcs, a supply of $b_1 = 23$ units of resource 1, and a supply of $b_2 = 12$ units of resource 2. Figure 3 provides the data for each potential undirected arc (p,q), $1 \le p < q \le 6$; the upper left corner of the cell corresponding to arc (p,q) contains the arc's index j; the center of the cell contains the cost c_j of including the arc in the spanning tree; the lower right corner contains the amounts a_{1j} and a_{2j} of resources 1 and 2, respectively, consumed by including the arc in the spanning tree. For example, the arc incident to nodes 3 and 5 has index 11 and, if included in the spanning tree, would cost 8.5 and consume 4 units of the first resource and 2 units of the second resource.

The application of the branch-and-bound algorithm (with $\beta = 0.6$ in Step 3 and an error tolerance of $\varepsilon = 0$ in Step 5) results in the bεb-tree of Figure 4. Each box (node) of the bεb-tree corresponds to a subproblem (P_{AB}) and contains, either explicitly or implicitly, the subproblems "vector of information" $[A,B,\lambda,w^*_{AB}(\lambda),s^*_{AB}(\lambda)]$ as described in the discussion of Steps 1, 2, and 10 (to conserve space, $w^*_{AB}(\lambda)$ and $s^*_{AB}(\lambda)$ are abbreviated by w and s, respectively); if a box does not explicitly contain all components of the "vector of information", it is implicit that the missing items are identical to the corresponding items in the box's "father" in the b&b-tree. The boxes are numbered according to the order in which they appeared during the execution of the algorithm; two boxes created simultaneously by partitioning (branching) in Step 7 have the same number and will be distinguished hereafter by appending "a" to the number of the left most box and "b" to the rightmost box. Thus, at "time" t ($1 \leq t \leq 18$), the b&b-tree consists of the subtree in Figure 4 that spans the boxes having numbers less than or equal to t; the leaves of this subtree comprise the stack of unfathomed subproblems at "time" t.

Box 1, the root of the b&b-tree, indicates that the minimal spanning problem that results from ignoring the resource constraints in (P) has an optimal solution that consumes 1 more unit of each resource than is available; hence, the branch-and-bound algorithm must be used. A box having only one branch leaving it (boxes 1, 3a, 3b, 5a, 5b,7a, 9a, 9b, 12a, and 17a) corresponds to the start of a sequence of iterations of subgradient optimization (Steps 3-6); the results of subgradient optimization appear immediately below in the box's only "son", where the son contains "no change" if subgradient optimization failed to improve the

lower bound $w^*_{AB}(\lambda)$; the intermediate results of subgradient optimization do not appear in Figure 4 but instead are summarized along the branch from father to son by a fraction (in parentheses) consisting of a denominator equal to the number of iterations and a numerator equal to the number of iterations that resulted in an improved lower bound. A box containing the results of a sequence of iterations of subgradient optimization is one of two types. The first type (boxes 2, 4, 6, 8, 11, 14, and 16) arises when subgradient optimization is terminated via Step 6 so that the subproblem (P_{AB}) must be partitioned in Step 7 into $(P_{A\cup\{j^*\},B})$ and $(P_{A,B\cup\{j^*\}})$, represented respectively in Figure 4 by the box's "left" son and "right" son; the second type (boxes 10, 13, and 18) arises when subgradient optimization terminates via Step 5 with the subproblem's fathoming (for the reason given in parentheses within the box). Boxes 7b, 12b, 15a, 15b, and 17b, the only boxes not yet discussed, each represent a subproblem created by a partitioning in Step 7 but never partitioned further, either because it (box 12b or 17b) was fathomed (for the reason given in parentheses within the box) in Step 8 or because just prior to the algorithm's termination it (box 7b, 15a, or 15b) was one of several fathomed simultaneously in Step 10 when the lower bound of the top item (box 15a) on the stack of unfathomed subproblems exceeded the incumbent's objective value.

As indicated by asterisks in Figure 4, there were three incumbent changes at Step C of the fathoming subroutine: the first between boxes 1 and 2 during a subgradient optimization iteration that failed to improve the lower bound $w^*_{AB}(\lambda)$ and the second and third in boxes 13 and 17b, respectively, when the subproblems they represent were fathomed due to

complementary slackness. Note that the third and last incumbent is optimal. The optimal spanning tree consists of arcs (1,6), (6,5), (1,2), (6,4), and (4,3); it has a total cost of 22.7 and consumes 23 units of the first resource and 12 units of the second resource.

This small example illustrates the potential advantages of a branch-and-bound approach based on Lagrangean relaxation. Although only 8 of the 49 iterations of subgradient optimization resulted in an improvement of at least 0.1% in the lower bound, these 8 increases in the lower bound together with the three incumbent changes resulted in a small b&b-tree. As discussed at the end of Section 6, a more traditional branch-and-bound approach that bounds the optimal objective value of (P_{AB}) only once by solving (P^0_{AB}) results in a b&b-tree having over 400 nodes.

4. SOLVING THE RCMST PROBLEM

If $e_j < \sum_{p=2}^{n} d_p$ for some arc j, then the optimal solution to (P) need not satisfy every arc capacity constraint. Consequently, the branch-and-bound algorithm of the previous section must be modified.

Recall that T denotes the finite set of incidence vectors corresponding to trees. Let F denote a subset of T such that $x \in F$ if and only if x corresponds to a tree in which there exists a feasible flow, that is, a flow from the source satisfying the demands at the sinks without violating any arc capacity constraints. The RCMST problem can now be expressed as

$$(Q) \qquad \text{Minimize} \quad \sum_{j=1}^{n} c_j x_j$$

$$\text{subject to} \quad \sum_{j=1}^{n} a_{ij} x_j \leq b_i \quad \text{for} \quad 1 \leq i \leq m$$

$$\text{and} \quad x \in F \ .$$

Given a set A consisting of indices of "admitted" arcs and a set B consisting of indices of "banished" arcs, define the subproblem (Q_{AB}) by

$$(Q_{AB}) \qquad \text{Minimize} \quad \sum_{j=1}^{n} c_j x_j$$

$$\text{subject to} \quad \sum_{j=1}^{n} a_{ij} x_j \leq b_i \quad \text{for} \quad 1 \leq i \leq m \ ,$$

$$\text{and} \quad x \in F_{AB} \ ,$$

where $F_{AB} = F \cap \{x \mid x_j = 1 \text{ for } j \in A\} \cap \{x \mid x_j = 0 \text{ for } j \in B\}$. Since $F_{AB} \subseteq T_{AB}$, the optimal objective value of (P_{AB}^{λ}) for any nonnegative λ furnishes a lower bound on the optimal objective value of (Q_{AB}); that is, the optimal objective value of (Q_{AB}) can be bounded from below by relaxing the resource constraints and ignoring the requirement that there exist a feasible flow in the tree corresponding to x. Ignoring rather than relaxing the requirement of a feasible flow is necessary because $x \in F_{AB}$ cannot be expressed as a system of linear inequalities in the binary vector x.

Overview of Necessary Modifications. Suppose that the subproblem (Q_{AB}) is under consideration at some point in the branch-and-bound algorithm. Recall that if $A \neq \phi$, the A-arcs (the arcs whose indices are

in A) form the A-subtree which includes the source as one of its nodes. Given the A-subtree and $j \in A$, let f_j denote the unique flow on arc j necessary if the source is to satisfy the demands of the other nodes in the A-subtree. For example, consider the A-subtree of Figure 5, where node 1 is the source, the demand d_p for each other node p is shown adjacent to the node, and the ordered pair adjacent to each arc j consists respectively of the capacity e_j of the arc and the unique flow f_j on the arc necessary if the source is to satisfy all demands. Assume inductively that $\{f_j \mid j \in A\}$ is known and is a component of the "vector of information" for the subproblem (Q_{AB}). Also assume inductively that instead of being stored in the "vector of information" as a set of arc indices, A is stored as an ordered list of ordered pairs of nodes, where an ordered pair (p,q) appears in the k-th position on the list if the undirected arc incident to nodes p and q was the k-th arc added to the A-subtree and the direction of flow on the arc is from p to q; note that the index of arc (p,q) can be computed easily whenever needed from

the formula given at the beginning of Section 1. For the example of Figure 5, suppose A is stored as the following ordered list of ordered pairs

Arc (p,q)	Index j
(1,11)	10
(11,4)	46
(1,6)	5
(11,12)	96
(12,15)	102
(11,7)	73
(12,2)	24

,

where the indicated index of each ordered pair (p,q) is not part of the data structure but is computed whenever needed from the formula given at the beginning of Section 1 (with N = 15); note that the direction of flow on each arc is from p to q. Given such a data structure for A, it is simple to update the flows when an arc is added to the A-subtree; the procedure would be as follows:

(1) Let (p,q) denote the arc to be added, where p is a node in the A-subtree and q is not; of course, the flow on arc (p,q) must be d_q;

(2) increase by d_q the flow on each arc of the unique path from the source to node p; note that this is easily done in one upward pass through the data structure for A since the ordered pair (r,s) that must precede the ordered pair (s,t) on the only path from the source to node

p is the unique ordered pair higher on the list in which s appears as the second component. In the example of Figure 5, suppose that an arc (2,q) is added to the A-subtree; then by repeatedly scanning the second component of the above data structure for A, the flows on the arcs of the unique path from the source to node 2 are increased by d_q in the reverse order (12,2), (11,12), and (1,11).

Given the above overview, it is now possible to describe the modifications to the branch-and-bound algorithm of Section 2 necessary for it to solve (Q) instead of (P), that is, the RCMST problem instead of the RMST problem.

Steps 1, 2, and 10. As discussed in the overview, the "vector of information" characterizing a subproblem (Q_{AB}) now includes the current values of the flows on the arcs of the A-subtree. In particular, the "vector of information" is $[A,B,\{f_j \mid j \in A\},\lambda,w^*_{AB}(\lambda),s^*_{AB}(\lambda)]$, where A is stored as the data structure discussed in the overview. Since $\{f_j \mid j \in A\}$ is uniquely determined by A, an alternative to including $\{f_j \mid j \in A\}$ in the "vector of information" is computing it whenever necessary; however, because the computation cannot be performed in a straightforward manner, the tradeoff between increased computation time and decreased storage does not seem to favor this alternative.

Step 3. No modifications are necessary.

Step 4. The feasibility check in Part B of the Fathoming Subroutine will be facilitated if, as the Prim-Dijkstra algorithm starts with the A-subtree and adds an arc at each iteration, the flows on the arcs of

each successive subtree are updated (using the procedure discussed in the overview) until either (1) an arc is added at some iteration that forces the flow on at least one arc of the current subtree to exceed its capacity or (2) the optimal solution to (P_{AB}^{λ}) is obtained and the flow on the corresponding tree satisfies the arc capacity constraints. Note that (1) may occur since the requirement that there exist a feasible flow in the optimal solution is not present in (P_{AB}^{λ}).

Step 5. The Fathoming Subroutine makes and attempt to fathom the subproblem (Q_{AB}) using the results of solving (P_{AB}^{λ}) for some $\lambda \geq 0$. Of course, the feasibility check in Part B now refers to checking whether $x_{AB}^{*}(\lambda)$ is a feasible solution of (Q_{AB}) and, therefore, (Q); the check is easy given $s_{AB}^{*}(\lambda)$ and the modification described in Step 4.

Step 6. No modifications are necessary.

Steps 7, 8, and 9. No modifications are necessary to Step 7; as before, j^{*} is computed and (Q_{AB}) is partitioned into the two subproblems $(Q_{A\cup\{j^{*}\},B})$ and $(Q_{A,B\cup\{j^{*}\}})$. The latter subproblem is treated as before in Steps 8 and 9; however, the former subproblem is handled differently. First, an attempt is made to fathom $(Q_{A\cup\{j^{*}\},B})$ by checking for the occurrence of one of the two situations described in the discussion of Step 8 in Section 3; then, if this is unsuccessful, the flows on the arcs of the $A\cup\{j^{*}\}$-subtree are updated (using the procedure discussed in the overview) to reflect the addition of the arc j^{*} to the A-subtree. If the updated flow violates at least one of the arc capacity constraints, $(Q_{A\cup\{j^{*}\},B})$

is fathomed because it has no feasible solution; otherwise, $(Q_{A\cup\{j^*\},B})$ is added to the stack of unfathomed subproblems in the form of the "vector of information $[A \cup \{j^*\}, B, \{f_j \mid j \in A \cup \{j^*\}\}, \lambda^I, w^*_{A\cup\{j^*\},B}(\lambda^I), s^*_{A\cup\{j^*\},B}(\lambda^I)]$.

5. SOLVING EXTENSIONS OF THE RCMST PROBLEM

This section discusses the modifications to the branch-and-bound algorithm necessary to analyze two extensions of the resource-constrained, capacitated minimal spanning tree problem: the existence of multiple arcs between each pair of nodes and the existence of multiple sources.

Multiple Arcs Between Each Pair of Nodes

In many practical situations, it is possible to construct an arc between a pair of nodes in a variety of ways, with each way having its own associated cost, capacity, and consumption of resources. For example, it may be possible to increase the capacity of an arc between a pair of nodes by simultaneously increasing the construction cost, increasing the consumption of some subset of resources, and decreasing the consumption of another subset of resources.

Although requiring additional notation to explain, the modifications to the branch-and-bound algorithm necessitated by the existence of multiple arcs between each pair of nodes do <u>not</u> require a significant increase in computational effort. More specifically, assume the indices 1, 2, ..., n of the potential arcs for the spanning tree resulted from an arbitrary indexing rather than from the method described at the beginning of Section 1; for each node pair (p,q) with $p < q$, let J_{pq} denote a set of arc indices such that $j \in J_{pq}$ if and only if arc j is incident to nodes p

and q. The only modification to the branch-and-bound algorithm occurs in Step 4, the solution of (P_{AB}^{λ}) for a given A, B, and λ; in particular, (P_{AB}^{λ}) is now solved as follows:

(1) Form the A-subtree.

(2) For each node pair (p,q) for which at least one node is not in the A-subtree, temporarily replace the multiple arcs incident to the two nodes by a single arc having an associated weight of $\min_{j \in J_{pq}-B}[c_j + \sum_{i=1}^{m} \lambda_i a_{ij}]$ if $J_{pq} - B \neq \phi$ or ∞ if $J_{pq} \subseteq B$.

(3) Starting with the A-subtree, use the Prim-Dijkstra algorithm as described by Prim [21] to solve the minimal spanning tree problem with the arc weights as computed in (2).

Note that the only "extra work" necessitated by multiple arcs between each pair of nodes is the computations described in (2); since the effect of (2) is to ignore all but the "cheapest" arc (as measured by the revised cost $c_j + \sum_{i=1}^{m} \lambda_i a_{ij}$) between a pair of nodes, there is no increase in the size of the minimal spanning tree problem that must be solved to determine the optimal spanning tree for (P_{AB}^{λ}).

Multiple Sources

Consider an extension of the problem in which each node must have access to one and only one supply center (hereafter, a source) from which it will receive a particular commodity. Suppose there are two means of creating access from a node to a source: (1) direct access, that is, construction of a source at the node itself, and (2) indirect access, that

is, construction of a unique path to the node from one and only one other node at which a source has been built. Clearly, the resulting network is a spanning forest, that is, a network in which each node belongs to one and only one of a set of disconnected trees; note that each tree in the spanning forest contains a single source and the (possibly empty) set of nodes it supplies in addition to itself. Restricting the network to a spanning forest implies that there are reasons (perhaps physical or technological) that prevent the flow destined for a particular node to be transmitted simultaneously from more than one source or along more than one path.

As before, if the spanning forest ultimately includes a potential undirected arc, the arc would have a known flow capacity and would result in the incurring of a known cost and the consumption of known quantities of each of the resources. Also as before, associated with each node is a known demand for the commodity; however, in addition, if a source were constructed at node p, it would have a known supply of e'_p units and would result in the incurring of a known cost c'_p and the consumption of a'_{ip} units of resource i, $1 \leq i \leq m$. Given the above quantities as well as the available supply of each resource used in the construction of arcs and/or sources, the problem is to construct a minimal cost spanning forest subject to the following requirements: (1) all resource constraints are satisfied, (2) each tree in the forest contains exactly one node at which a source has been constructed, and (3) there exists a feasible flow in each tree in the forest. Note that a resource constraint cannot only be used (as indicated in Section 1) to limit the degree of a node in the spanning forest but can also be used to limit the number of sources constructed.

A straightforward procedure transforms this minimal spanning forest problem into an RCMST problem; more specifically, augment the original N-node network with a super-source, a new node having an infinite supply of the commodity, and join each node p to the super-source with an artificial arc having a capacity of e'_p, a cost of c'_p, and resource consumptions a'_{ip}, $1 \leq i \leq m$. Thus, the data associated with the construction of a facility at node p is now associated with the artificial arc joining node p with the super-source. Note that multiple artificial arcs between a node and the super-source could be used (as discussed earlier in this section) if there were more than one way to construct a source at node p. Clearly, the optimal spanning tree in the revised network is equivalent to the optimal spanning forest in the original network; in particular, the optimal spanning forest for the original problem is obtained from the optimal spanning tree for the revised problem by deleting the super-source and any arc incident to it, with the understanding that a source is constructed at each node to which a deleted artificial arc was incident. Thus, this multiple-source version of the problem is equivalent to an RCMST problem with one more node than the original problem; as such, its solution entails a negligible increase in computational effort over the RCMST problem.

6. COMPUTATIONAL EXPERIENCE AND CONCLUSIONS

A FORTRAN computer program for solving the RCMST problem has been implemented on the IBM 3033 computer at Stanford University. The computer program consists of approximately 1000 statements. The variable memory requirements of the program are a function of the number of nodes in the

network, the number of resource constraints, and the maximum number of unfathomed subproblems that may be on the stack simultaneously; denote these parameters by N, m, and S, respectively. The required total number of kilobytes (1000 bytes) of memory is given by

$$\left[\frac{8N^2+22N-20}{1000}\right] + \left[\frac{2mN(N-1)}{1000}\right] + \left[\frac{(N+96)(N-1)}{16} + (10+8m)\right]\left[\frac{S}{1000}\right] .$$

Table I tabulates the above expression as a function of S for various values of N and m; the table is approximate in the sense that the constant term has been rounded to one decimal place and the coefficient of S has been rounded to three decimal places. It can be seen from both the expression and the table that the memory requirement is most affected by S; the last column of Table I provides the maximum value of S assuming an availability of at most 7000 kilobytes of memory, the approximate availability of memory on Stanford's IBM 3033. Table I, in conjunction with the computational experience described below, indicates that in most cases the limiting factor in solving RCMST problems should be the execution time rather than the memory requirement.

In order to gain some preliminary computational experience with the algorithm, a random-problem generator was constructed; it executes the following steps:

(1) The determination of the demands at the nodes is via one of three methods, depending on which problem type is desired. To obtain the most general RCMST problem, the demand d_p at node p ($2 \leq p \leq N$) is an independent random variable uniformly distributed on the integers in the interval [1, 50]. To obtain an important special case of the RCMST

problem, d_p is set equal to 1 for $2 \leq p \leq N$. Finally, to obtain an RMST problem (i.e., without flow constraints), d_p is set equal to 0 for $2 \leq p \leq N$.

(2) To insure the existence of at least one feasible solution to the RCMST problem, a feasible tree is randomly generated in N-1 iterations. More specifically, the trivial subtree consisting only of node 1 is iteratively expanded; iteration k consists of adding arc (p,q) to the subtree, where p is randomly selected from the k nodes belonging to the current subtree and q is randomly selected from the N-k nodes not yet a member of the current subtree. Let T_r denote the set of indices of those arcs comprising the randomly generated trees and, for $j \in T_r$, let f_j denote the flow on arc j that would be necessary to satisfy the node demands. Then, for $j \in T_r$, the flow capacity e_j is an independent random variable uniformly distributed on the integers in the interval $[f_j, \sum_{p=2}^{n} d_p]$; note that this insures that the randomly generated tree will always satisfy the flow capacity constraints. Also, for $j \in T_r$, the construction cost c_j is an independent random variable uniformly distributed on the integers in the interval [0, 99], and the consumption a_{ij} of resource i ($1 \leq i \leq m$) is an independent random variable uniformly distributed on the integers in the interval [0,9].

(3) Recall that there are potentially $n = (1/2)(N)(N-1)$ undirected arcs, each with a distinct index j equal to its position in a lexicographic ordering of the n arcs based on the representation of an arc as a vector (p,q) with $p < q$. For each arc j that is <u>not</u> a member of the randomly generated tree, an independent random number ρ_j is generated that is uniformly distributed over the continuous interval

[0,1]. If $\rho_j \leq (10/N)$, then arc j is referred to as a "potential arc"; the associated c_j and a_{ij} $(1 \leq i \leq m)$ are randomly generated as described in Step 2, but the flow capacity e_j is an independent random variable uniformly distributed on the integers in the interval $[\underset{2 \leq p \leq n}{\text{minimum}}\ d_p,\ \sum_{p=2}^{n} d_p]$. If $\rho_j > (10/N)$, then arc j is effectively eliminated from inclusion in the optimal spanning tree because c_j and a_{ij} $(1 \leq i \leq m)$ are set equal to an extremely large number and e_j is set equal to zero. The usage of $(10/N)$ as the "cutoff valve" for ρ_j implies that, for $n \geq 10$, the expected number of potential arcs for inclusion in the spanning tree is $5(N-1) + \Delta$, where $0 < \Delta < (N-1)$ reflects the special treatment given to the arcs of the randomly generated tree; thus between $5N$ and $6N$ is a reasonable estimate for the number of potential arcs in any randomly generated problem.

(4) The remaining step is to determine values for b, an m-vector whose i-th component is the availability b_i of resource i $(1 \leq i \leq m)$. Let b_L denote an m-vector whose i-th component is the amount of resource i consumed by the randomly generated spanning tree of Step 2. Also, let b_U denote an m-vector whose i-th component is the maximum amount of resource i consumed by <u>any</u> spanning tree comprised of any of the potential arcs; this maximal usage of resource i is found by solving a maximal spanning tree problem where the "cost" of arc j is a_{ij}. Finally, for some constant λ satisfying $0 \leq \lambda \leq 1$, let $b = (1-\lambda)b_L + \lambda b_U$. Note with this choice of b that the randomly generated spanning tree of Step (2) satisfies the resource constraints for all values of λ; also note that as λ increases from 0 to 1, the number of spanning trees satisfying the resource constraints also

increases. Rather than use $b = (1-\lambda)b_L + \lambda b_U$ for some $0 \leq \lambda \leq 1$, a slight modification is made. More specifically, to insure a nontrivial solution to the RCMST problem, it is necessary to insure the infeasibility of the spanning tree that solves the simple MST problem with the arc costs of the RCMST problem. To do so, this simple MST problem is solved, some resource i^* is chosen for which the consumption of resource i^* in the MST equals a value v that is strictly greater than component i^* of b_U, and component i^* of b_U is reset to v. (Note that the assumption of the existence of such an i^* is made without loss of generality since such a condition can be obtained eventually by simply discarding the current set of data and returning to Steps (2) and (3) to generate new data.) After resetting component i^* of b_U, recomputing $b = (1-\lambda)b_L + \lambda b_U$ $(0 \leq \lambda < 1)$ leads to a set of resource constraints that is always satisfied by the randomly generated spanning tree of Step (2) but never satisfied by the solution to the simple MST problem; thus, a nontrivial optimal solution to the RCMST problem exists. In the computational experience discussed below, $\lambda = 0.5$ was used.

Tables II, III, and IV summarize the results of some of the computational experience. Table II summarizes results for six RMST problems each having 50 nodes and 5 resource constraints; Table III summarizes the results for six RCMST problems each having 20 nodes, 3 resource constraints, and a unit demand at every node; and Table IV summarizes the results for six RCMST problems having 20 nodes, 3 resource constraints, and random node demands as described in Step (1) of the random-problem generator. The column headings of Table II are described more explicitly by the following:

(1) The second column contains 100ε. Recall that 100ε is the maximum percentage error that might result from using the fathoming rules discussed in Section 3; that is, the cost of the "ε-optimal" solution to the problem may be at most 100ε percent higher than the exact ($\varepsilon=0$) minimal cost. The actual percentage error that resulted is given in the ninth and last column of Table II.

(2) The third column contains the number of nodes in the branch-and-bound tree generated to solve the problem.

(3) The fourth column contains the number of MST problems solved during the executions of Steps 4 and 8 of the algorithm.

(4) The fifth column contains the maximum number of unfathomed subproblems that were simultaneously on the stack and, in conjunction with Table I, provides insight into the memory requirements of the problem.

(5) The sixth, seventh and eighth columns are related. The sixth column (labeled (a)) contains the CPU time (excluding compilation) that elapsed until the eventual ε-optimal solution became the incumbent solution, and the seventh column (labeled (b)) contains the total CPU time (excluding compilation) required to solve the problem. The ratio $100(b-a)/b$ contained in the eighth column is the percentage of the total execution time that was required to verify that the ε-optimal solution that became the incumbent at the time given in the column labeled (a) was, in fact, an ε-optimal solution; in this sense, the ratio measures "wasted" computational effort. The above descriptions also apply to the headings of the columns of Tables III and IV.

The purpose of the computational experience summarized by Tables II, III, and IV was to demonstrate the algorithm's potential to solve large problems and to gain insight into the algorithm's behavior while solving a series of related problems. The reported execution times are meaningful only in this context. Lower (Higher) average execution times for similar problems could have been obtained by decreasing (increasing) the arc density in Step (3) of the random-problem generator and/or by increasing (decreasing) λ in Step (4) of the random-problem generator. Also, significantly lower execution times would result if a computer programmer more knowledgeable than this author were to implement a more sophisticated algorithm (c.f., [4], [19], or [24]) for solving the MST problems in Steps (4) and (8) of the algorithm; the solution of the MST problems constitutes a major portion of the computational effort so that a more efficient algorithm would have a significant favorable impact on execution times. Another operation done repetively during the algorithm's execution is the insertion of a subproblem into its proper position on the stack of unfathomed subproblems based on the value of its lower bound; currently done in a somewhat naive manner, this operation could be coded by a more sophisticated computer programmer than this author to execute significantly faster. A final potential way to reduce execution time is to "fine-tune" the parameters used to revise λ in Step (3) of the algorithm as described in Section 3.

In light of the comments of the preceding paragraph, conjectures rather than conclusions are in order. Table II indicates that the algorithm has the potential to solve very large RMST problems, especially if one is willing to forego exact optimality and accept accuracy to within

1% or even 5%. However, Tables III and IV indicate that the algorithms potential for solving large RCMST problems is less promising, even if one is willing to accept accuracy to within 5% rather than exact optimality. However, since the i-th problem of Table III is identical to the i-th problem of Table IV with the exception of node demands, it does appear that the RCMST problem with unit demands is significantly easier to solve than the general RCMST problem.

One final point should be made. It is possible to "trick" the computer program into the more traditional branch-and-bound approach that bounds a subproblem (P_{AB}) by solving only the relaxation (P^0_{AB}), that is, the MST problem that results from ignoring the resource constraints. Thus, the traditional approach and the Lagrangean-based approach can be compared with the same program on the computer. Computational experience indicates that a traditional branch-and-bound approach to the RMST and RCMST problems fails due to a lack of sufficient computer memory; because of the frequent branching of the traditional approach, the size of the list of unfathomed subproblems grows so rapidly that available computer memory is rapidly exceeded. This is illustrated by the example at the end of Section 3 where the branch-and-bound algorithm based on Lagrangean relaxation resulted in a b&b-tree with only 18 nodes as compared to one with over 400 nodes resulting from the traditional approach.

REFERENCES

[1] Bradley, G. H., Survey of Deterministic Networks, AIIE Trans. 7 (1975), 222-234.

[2] Camerini, P. M., L. Fratta, and F. Maffioli, On Improving Relaxation Methods by Modified Gradient Techniques, Math. Prog. Study 3 (1975), 26-34.

[3] Chandy, K. M., and T. Lo, The Capacitated Minimum Spanning Tree, Networks 3 (1973), 173-181.

[4] Cheriton, D., and R. E. Tarjan, Finding Minimum Spanning Trees, SIAM J. Comput. 5 (1976), 724-742.

[5] Crowder, H. P., Computational Improvements for Subgradient Optimization, Symposia Mathematica 19 (1976), 357-372.

[6] Dijkstra, E. W., A Note on Two Problems in Connection with Graphs, Numerishe Mathematik 1 (1959), 269-271.

[7] Duke, C. M., and D. F. Moran, Earthquakes and City Lifelines, San Fernando Earthquake of February 9, 1971 and Public Policy, a report to the Special Subcommittee of the Joint Committee on Seismic Safety - California Legislature (1971).

[8] Edmonds, J., Matroids and the Greedy Algorithm, Math. Prog. 1 (1971), 127-136.

[9] Fisher, M. L., The Lagrangian Relaxation Method for Solving Integer Programming Problems, Man. Sci. 27 (1981), 1-18.

[10] _____, W. D. Northup, and J. F. Shapiro, Using Duality to Solve Discrete Optimization Problems: Theory and Computational Experience, Math. Prog. Study 3 (1975), 56-94.

[11] Gavish, B., Formulations and Algorithms for the Capacitated Minimal Directed Tree Problem, Working Paper No. 8016, Graduate School of Management, University of Rochester, Rochester, N.Y. (1980).

[12] _____, Topological Design of Centralized Computer Networks - Formulations and Algirithms, Working Paper No. 8009, Graduate School of Management, University of Rochester, Rochester, N.Y. (1981).

[13] Geoffrion, A. M., Lagrangean Relaxation for Integer Programming, Math. Prog. Study 2 (1974), 82-114.

[14] Held, M., and R. M. Karp, The Traveling Salesman Problem and Minimum Spanning Trees, Opns. Res. 18 (1970), 1138-1162.

[15] _____ and _____, The Traveling Salesman Problem and Minimum Spanning Trees: Part II, Math. Prog. 1 (1971), 6-25.

[16] _____, P. Wolfe, and H. P. Crowder, Validation of Subgradient Optimization, Math. Prog. 6 (1974), 62-88.

[17] Hillier, F. S., and G. J. Lieberman, Introduction to Operations Research, Holden-Day, Inc., San Francisco (1980).

[18] Kershenbaum, A., Computing Capacitated Minimal Spanning Trees Efficiently, Networks 4 (1974), 299-310.

[19] _____ and R. Van Slyke, Computing Minimum Spanning Trees Efficiently, Proc. 25th Ann. Conf. of the ACM (1972), 518-527.

[20] Kruskal, J. B., On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem, Proc. Amer. Math Soc. 7 (1956), 48-50.

[21] Prim, R. C., Shortest connection Networks and Some Generalizations, Bell Systems Tech. J. 36 (1957), 1389-1401.

[22] Rothfarb, B., H. Frank, D. M. Rosenbaum, and K. Steiglitz, Optimal Design of Offshore Natural-Gas Pipeline Systems, Opns. Res. 18 (1970), 992-1020.

[23] Shapiro, J. F., A survey of Lagrangian Techniques for Discrete Optimization, Ann. Discrete Math. 5 (1979), 113-138.

[24] Yao, A. C., An $O(|E| \log \log |V|)$ Algorithm for Finding Minimum Spanning Trees, Information Processing Letters, 4 (1975), 21-23.

Table I

Memory Requirements for Computer Program

N	m	Approximate Memory Requirement (Kilobytes)	Maximum Value of S Assuming Availability of 7000 Kilobytes of Memory
10	5	1.9 + 0.110S	63,619
20	5	7.4 + 0.188S	37,194
30	5	16.5 + 0.278S	25,120
40	5	29.3 + 0.382S	18,247
50	5	45.6 + 0.497S	13,992
10	5	1.9 + 0.110S	63,619
20	10	11.2 + 0.228S	30,652
30	15	33.9 + 0.358S	19,458
40	20	76.1 + 0.502S	13,792
50	25	143.6 + 0.657S	10,435
10	10	2.8 + 0.150S	46,648
20	20	18.8 + 0.308S	22,666
30	30	60.0 + 0.478S	14,518
40	40	138.5 + 0.662S	10,364
50	50	266.1 + 0.857S	7,857

Table II

Computational Results for RMST Problems with 50 Nodes and 5 Resource Constraints

Problem Number	100ε	Number of Nodes in B&B-Tree	Number of MST Problems Solved	Maximum Stack Size	Execution Time (seconds) Until ε-optimal Solution (a)	Total Execution Time (seconds) (b)	Ratio 100(b-a)/b	Actual Percentage Error
1	0	313	937	12	1.06	21.02	95.0	0.0
	1	13	37	6	1.05	1.05	0.0	0.0
	5	3	7	1	0.15	0.19	21.1	3.2
2	0	5	13	2	0.26	0.37	29.7	0.0
	1	3	8	2	0.26	0.26	0.0	0.0
	5	3	7	1	0.13	0.20	35.0	3.0
3	0	465	1395	10	0.21	31.27	99.3	0.0
	1	1	8	1	0.21	0.21	0.0	0.0
	5	1	8	1	0.21	0.21	0.0	0.0
4	0	243	735	12	1.64	16.82	90.2	0.0
	1	17	59	3	1.47	1.47	0.0	0.0
	5	3	15	1	0.17	0.38	55.3	4.2
5	0	359	1075	10	5.04	24.36	79.3	0.0
	1	3	12	1	0.16	0.34	52.9	0.6
	5	1	5	1	0.16	0.16	0.0	0.6
6	0	677	2031	7	2.93	46.17	93.7	0.0
	1	1	3	1	0.11	0.11	0.0	0.2
	5	1	3	1	0.11	0.11	0.0	0.2

Table III

Computational Results for RCMST Problems with Unit Demands, 20 Nodes, and 3 Resource Constraints

Problem Number	100ε	Number of Nodes in B&B-Tree	Number of MST Problems Solved	Maximum Stack Size	Execution Time (seconds) Until ε-optimal Solution (a)	Total Execution Time (seconds) (b)	Ratio 100(b-a)/b
1	5	1,477	4,830	571	22.51	22.51	0.0
2	5	1,261	4,017	412	18.83	18.83	0.0
3	5	69	441	25	1.80	1.90	5.3
4	5	113	415	48	1.90	1.90	0.0
5	5	19	60	9	0.29	0.29	0.0
6	5	11,351	38,271	3,021	175.16	175.16	0.0

Table IV

Computational Results for RCMST with Random Demands, 20 Nodes, and 3 Resource Constraints

Problem Number	100ε	Number of Nodes in B&B-Tree	Number of MST Problems Solved	Maximum Stack Size	Execution Time (seconds) Until ε-optimal Solution (a)	Total Execution Time (seconds) (b)	Ratio (b-a)/b
1	5	12,347	41,451	3,048	187.39	187.39	0.0
2	5	4,539	14,363	617	27.17	63.70	57.3
3	5	215	936	90	3.94	4.26	7.5
4	5	1,143	4,295	368	18.96	18.96	0.0
5	5	135	428	59	2.10	2.10	0.0
6	5	10,417	35,266	2,755	159.37	159.37	0.0

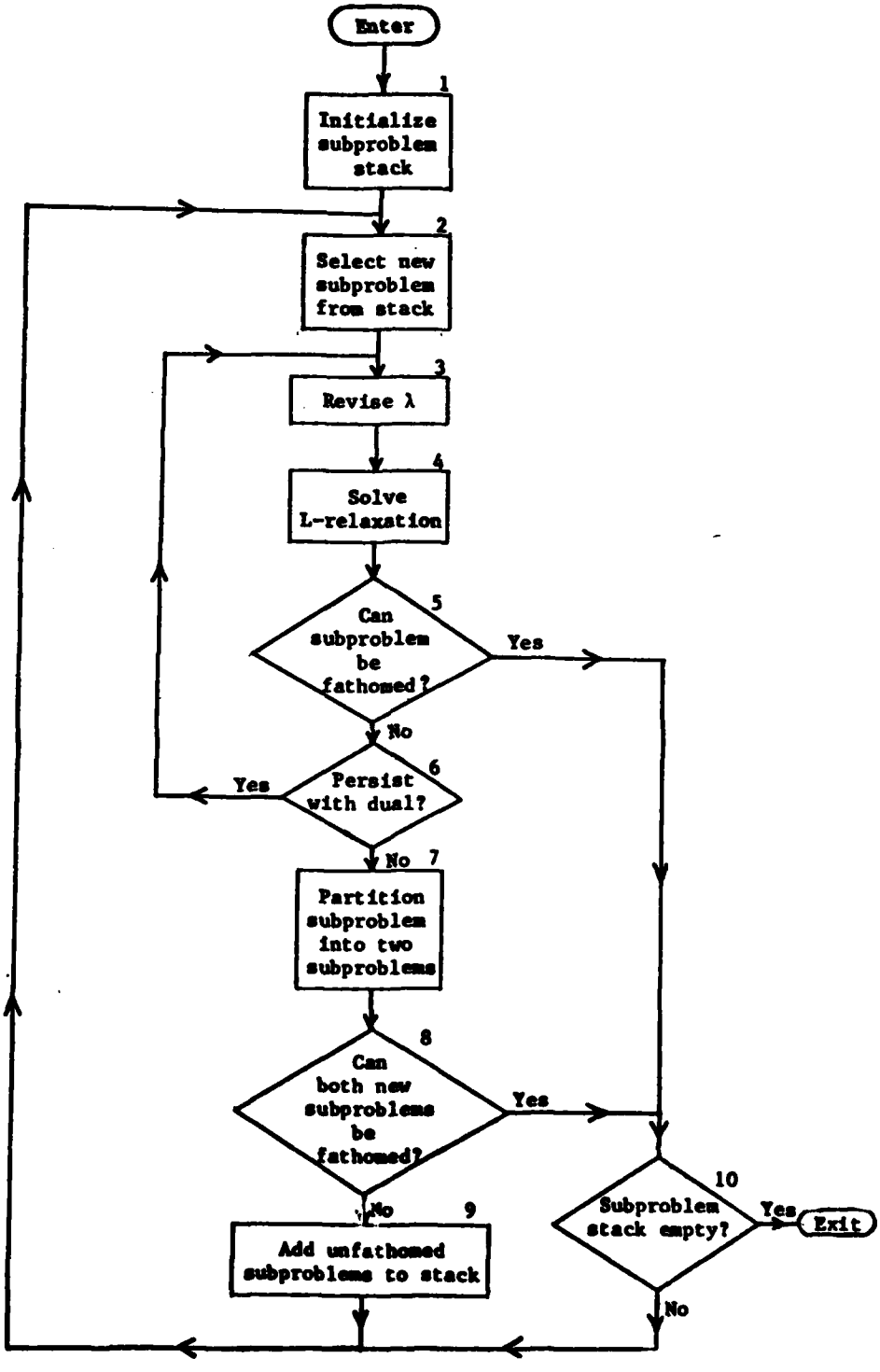

Enter
1
Initialize subproblem stack
2
Select new subproblem from stack
3
Revise λ
4
Solve L-relaxation
5
Can subproblem be fathomed?
Yes
No
6
Persist with dual?
Yes
No
7
Partition subproblem into two subproblems
8
Can both new subproblems be fathomed?
Yes
No
9
Add unfathomed subproblems to stack
10
Subproblem stack empty?
Yes
Exit
No

Figure 1

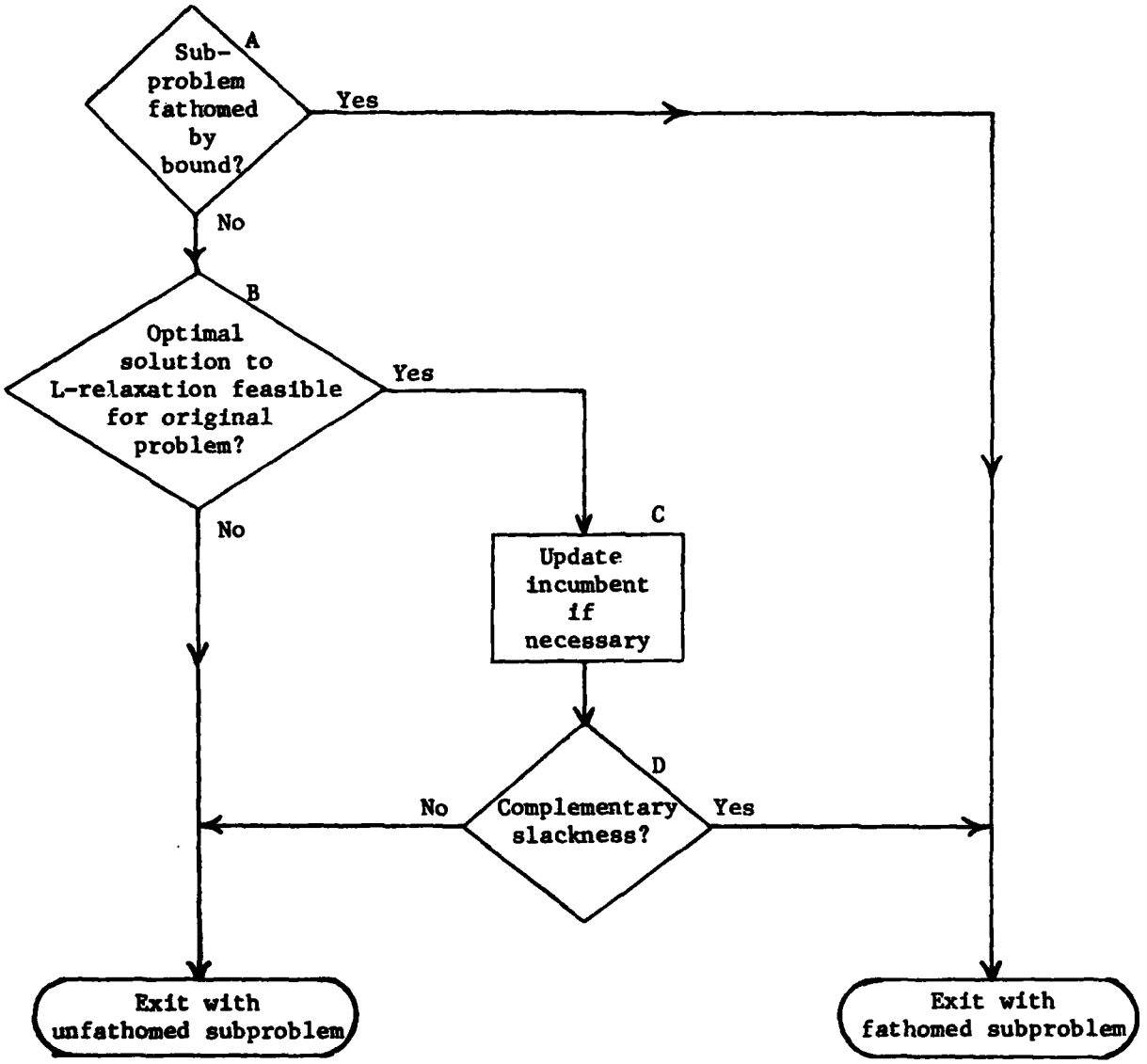

A
Sub-problem fathomed by bound?
Yes
No
B
Optimal solution to L-relaxation feasible for original problem?
Yes
No
C
Update incumbent if necessary
D
Complementary slackness?
No
Yes
Exit with unfathomed subproblem
Exit with fathomed subproblem

Figure 2

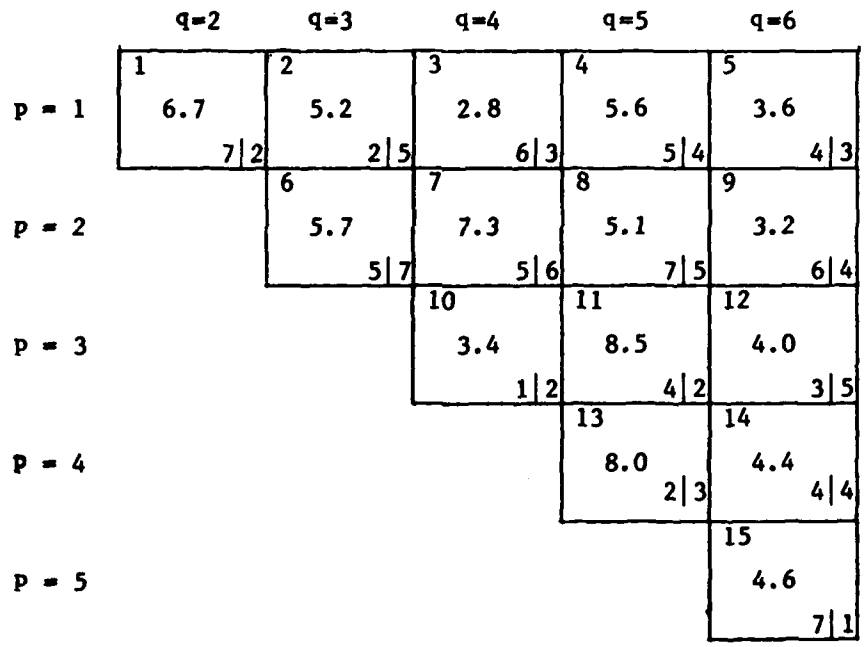

q=2
q=3
q=4
q=5
q=6
p = 1
p = 2
p = 3
p = 4
p = 5
1
6.7
7|2
2
5.2
2|5
3
2.8
6|3
4
5.6
5|4
5
3.6
4|3
6
5.7
5|7
7
7.3
5|6
8
5.1
7|5
9
3.2
6|4
10
3.4
1|2
11
8.5
4|2
12
4.0
3|5
13
8.0
2|3
14
4.4
4|4
15
4.6
7|1

Figure 3

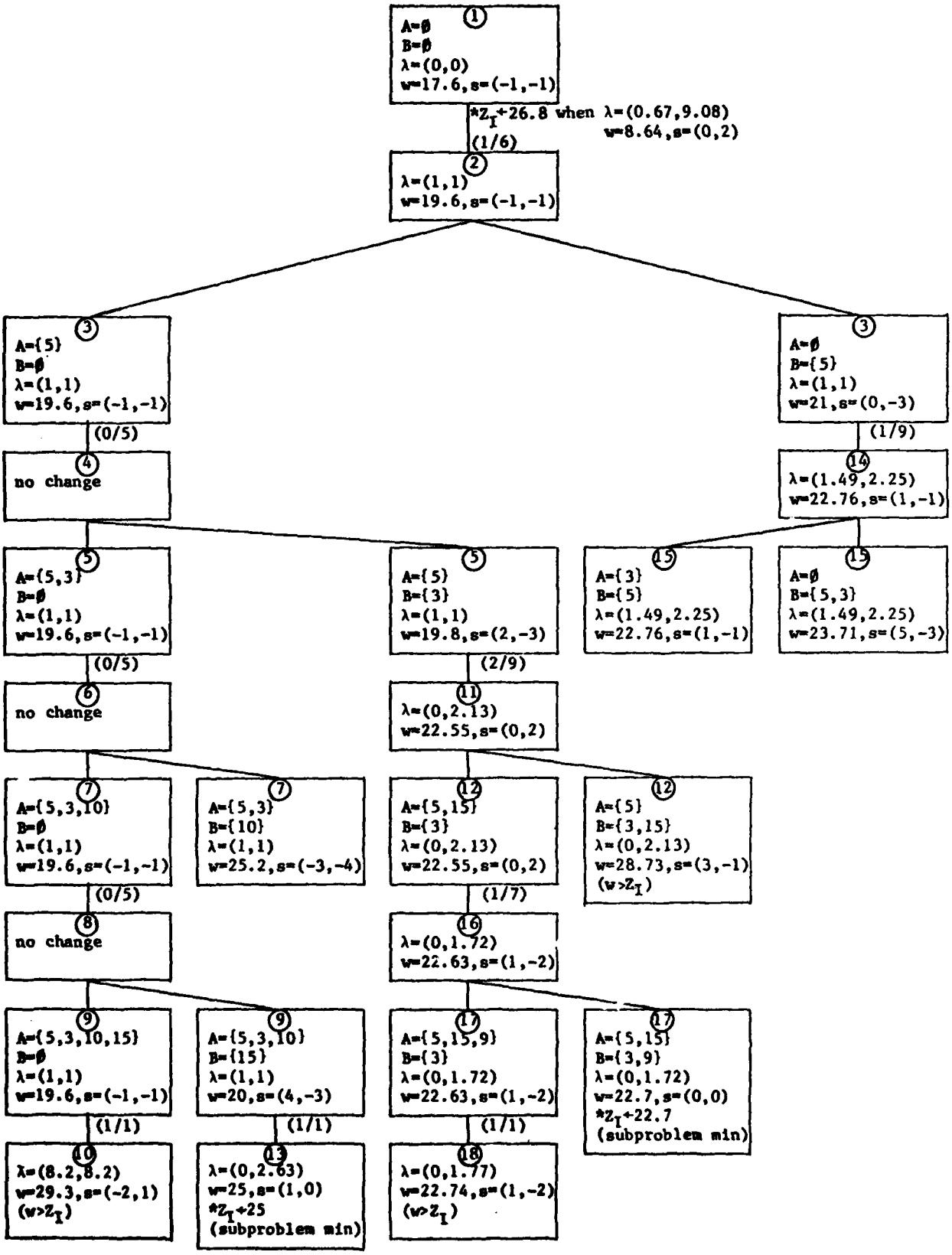

1
A=∅
B=∅
λ=(0,0)
w=17.6,s=(-1,-1)
*Z_I=26.8 when λ=(0.67,9.08)
w=8.64,s=(0,2)
(1/6)
2
λ=(1,1)
w=19.6,s=(-1,-1)
3
A={5}
B=∅
λ=(1,1)
w=19.6,s=(-1,-1)
(0/5)
4
no change
5
A={5,3}
B=∅
λ=(1,1)
w=19.6,s=(-1,-1)
(0/5)
6
no change
7
A={5,3,10}
B=∅
λ=(1,1)
w=19.6,s=(-1,-1)
(0/5)
8
no change
9
A={5,3,10,15}
B=∅
λ=(1,1)
w=19.6,s=(-1,-1)
(1/1)
10
λ=(8.2,8.2)
w=29.3,s=(-2,1)
(w>Z_I)
7
A={5,3}
B={10}
λ=(1,1)
w=25.2,s=(-3,-4)
9
A={5,3,10}
B={15}
λ=(1,1)
w=20,s=(4,-3)
(1/1)
13
λ=(0,2.63)
w=25,s=(1,0)
*Z_I=25
(subproblem min)
5
A={5}
B={3}
λ=(1,1)
w=19.8,s=(2,-3)
(2/9)
11
λ=(0,2.13)
w=22.55,s=(0,2)
12
A={5,15}
B={3}
λ=(0,2.13)
w=22.55,s=(0,2)
(1/7)
16
λ=(0,1.72)
w=22.63,s=(1,-2)
17
A={5,15,9}
B={3}
λ=(0,1.72)
w=22.63,s=(1,-2)
(1/1)
18
λ=(0,1.77)
w=22.74,s=(1,-2)
(w>Z_I)
12
A={5}
B={3,15}
λ=(0,2.13)
w=28.73,s=(3,-1)
(w>Z_I)
17
A={5,15}
B={3,9}
λ=(0,1.72)
w=22.7,s=(0,0)
*Z_I=22.7
(subproblem min)
3
A=∅
B={5}
λ=(1,1)
w=21,s=(0,-3)
(1/9)
14
λ=(1.49,2.25)
w=22.76,s=(1,-1)
15
A={3}
B={5}
λ=(1.49,2.25)
w=22.76,s=(1,-1)
15
A=∅
B={5,3}
λ=(1.49,2.25)
w=23.71,s=(5,-3)

Figure 4

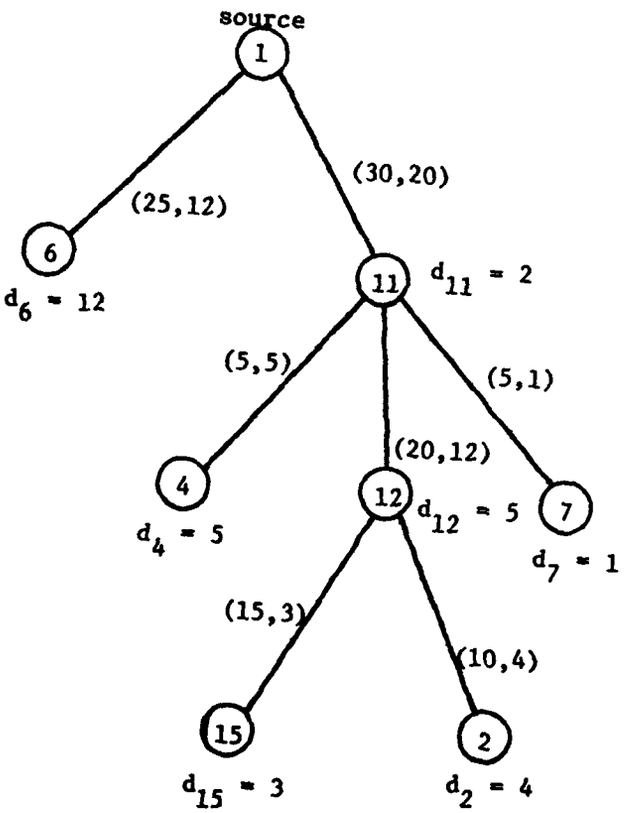

source
1
(25,12)
(30,20)
6
$d_6 = 12$
11
$d_{11} = 2$
(5,5)
(5,1)
(20,12)
4
$d_4 = 5$
12
$d_{12} = 5$
7
$d_7 = 1$
(15,3)
(10,4)
15
$d_{15} = 3$
2
$d_2 = 4$

Figure 5

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER 200	2. GOVT ACCESSION NO. AD-A115 354	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) CONSTRUCTING A MINIMAL COST SPANNING TREE SUBJECT TO RESOURCE CONSTRAINTS AND FLOW REQUIREMENTS		5. TYPE OF REPORT & PERIOD COVERED TECHNICAL REPORT 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) ANDREW W. SHOGAN		8. CONTRACT OR GRANT NUMBER(s) N00014-75-C-0561
9. PERFORMING ORGANIZATION NAME AND ADDRESS DEPT'S. OF OPERATIONS RESEARCH AND STATISTICS STANFORD UNIVERSITY STANFORD, CALIFORNIA 94305		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS NR-047-200
11. CONTROLLING OFFICE NAME AND ADDRESS OPERATIONS RESEARCH, CODE 434 OFFICE OF NAVAL RESEARCH ARLINGTON, VIRGINIA 22217		12. REPORT DATE NOVEMBER 1981 13. NUMBER OF PAGES 53
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

APPROVED FOR PUBLIC RELEASE: DISTRIBUTION IS UNLIMITED

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

NETWORK DESIGN
INTEGER PROGRAMMING
CONSTRAINED SPANNING TREE
BRANCH AND BOUND ALGORITHM
LAGRANGEAN RELAXATION

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

PLEASE SEE OTHER SIDE

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

Abstract REPORT NO. 200

Consider a network in which one node is a source having an infinite supply of a commodity, and every other node is a sink having a known constant demand. Furthermore, associated with each potential arc of the network are the following known constants: the cost of constructing the arc, the amount of each scarce resource consumed during the construction of the arc, and the flow capacity of the arc. Given the above known constants as well as the available supply of each of the scarce resources, the problem is to construct a minimal cost spanning tree subject to the limitations on the consumption of the scarce resources and the requirement that there exists a flow from the source satisfying the demands at the sinks without exceeding any arc capacity. This paper discusses the solution of such a problem by a branch-and-bound algorithm base on Lagrangean relaxation. Also included are applications of the problem, extensions to the problem, and a report on preliminary computational experience with a computer implementation of the algorithm.


DATE
ILME


DATE
ILME